Information Retrieval
Assignment-I

Rishabh Deshmukh(13349)

January 2017

1 Introduction

This is the report for the first assignment of this course in which i have implemented some basic functionality of a search engine using **Lucene 6.3.0** library. This assignment includes the implementation of **Indexing** of medium-sized corpus of documents related to Category: **Science and Technology in India** crawled from wikipedia followed by **Searching** on the same corpus by using the index table created during the process of indexing.

1.1 Indexing

Indexing is the process of building an index table of the corpus of documents. Index table consist of a Dictionary and Postings. Indexing involves various functions like tokenization, stemming, stop-word elimination, normalization etc. It assigns a unique number (called docId) to each document of the corpus. Each term in dictionary has a posting list of documents in which it occurs.

1.2 Searching

It is the process of retrieving documents from the corpus which are relevant for the given query. A query may have single or multiple terms and generally it needs some preprocessing before we can start searching for the documents. Query parser performs the preprocessing of input query.

2 Building Indexer

2.1 Data Acquisition

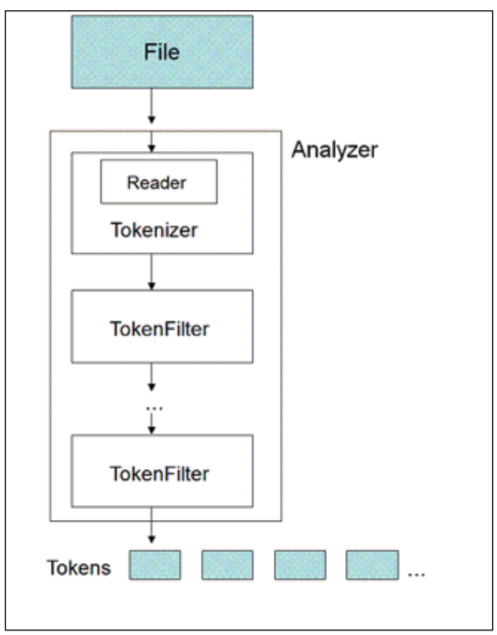
I have crawled wikipedia to acquire documents for the **Category:Science and Technology in India**. Documents were downloaded in .json (javascript object notation) format. These files needs to be parsed in order to get the relevant information.

2.2 Parsing of .json files

I have used **JSONParser** and **JSONObject** class from **org.json.simple package** to parse .json files to extract the relevant information. This is how I get the documents for indexing.

2.3 Analysis of Documents

Analysis involves parsing of documents to generate Stream of Tokens. Further processing of this Stream generates the terms that will be the part of dictionary. A typical analyzer contains one **Tokenizer** and one or more **TokenFilters**.



In my work I have used EnglishAnalyzer to analyze the documents. It make use of StandardTokenizer to tokenize the documents. It uses various filters for postprocessing of TokenStream generated by StandardTokenizer. StandardFilter, LowercaseFilter, StopwordFilter, PortemStemFilter etc are some of the filters used by EnglishAnalyzer. It also performs the task of normalization. In Lucene 6.3.0, EnglishAnalyzer class is final so I copied the whole code from EnglishAnalyzer and pasted into MyAnalyzer (custom class for implementing my own analyzer) so that I can add/remove various filters to evaluate the performance of Searching and Indexing.

2.4 Indexing

For indexing **Lucene** provides a class named **IndexWriter**. By using **addDocument()** function of this class we can add a document to the index. Argument of addDocument() is of Document type. So we need to create a Document object and need to add the relevant information extracted from .json file to the newly created Document object. At last, call the close() method on IndexWriter object to commit the changes.

3 Building Searcher

Lucene 6.3.0 library provides **IndexSearcher** class using which we can instantiate the searcher by providing **Reader** object of the directory in which the index is stored. Lucene also provide **QueryParser** class to parse the query. I have used the same parser in Indexing as well as in QueryProcessing so that same filters are used in both the cases. For similarity measure i have used default ranking mechanism of **Lucene** (i.e. BM25Similarity)

3.1 BM25Similarity Ranking Measure [Ref:Wikipedia]

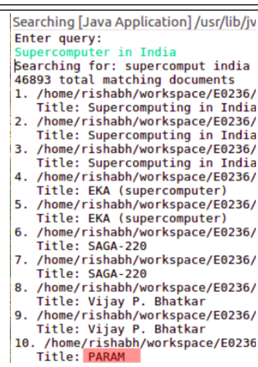
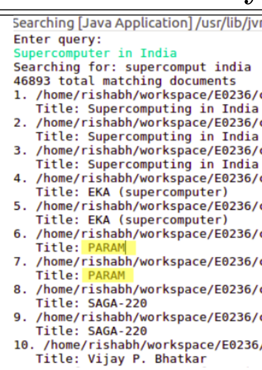
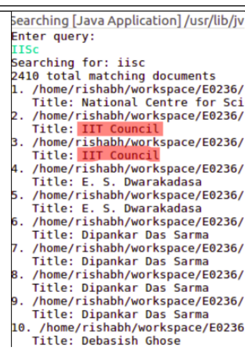
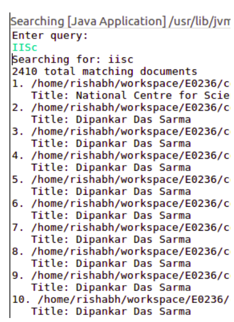
It is the Default Ranking Measure in **Lucene 6.3.0**. I have verified this by calling **getSimilarity()** method on IndexWriterConfig object as well as on IndexSearcher object.
BM25 is a bag-of-words ranking function (i.e. it doesn't consider the relative order and relative proximity between the query terms

that are present in the document).

Consider a query **Q** containing n terms $q_1, q_2, ..., q_n$.
The BM25 score of a matching document **D** can be computed using the following formula:

$$score(\mathbf{D},\mathbf{Q}) = \sum_{i=1}^{n} \frac{IDF(q_i).tf(q_i, \mathbf{D}).(k_1 + 1)}{tf(q_i, \mathbf{D}) + k_1.(1 - b + b.\frac{|\mathbf{D}|}{avgdl})}$$

where:
$IDF(q_i)$: Inverse Document Frequency of term q_i.
$tf(q_i, \mathbf{D})$: Term Frequency of term q_i in document **D**.
$|D|$: Length of document **D** in words.
$avgdl$: Average document length in the corpus from which document is drawn.
k_1 and b are free parameters. Where $k \in [1.2, 2]$ and $b = 0.75$.

<table>
<tr><th colspan="2">Comparison</th></tr>
<tr><th>ClassicSimilarity</th><th>BM25Similarity</th></tr>
<tr><td>
Searching [Java Application] /usr/lib/jv
Enter query:
Supercomputer in India
Searching for: supercomput india
46893 total matching documents
1. /home/rishabh/workspace/E0236/
Title: Supercomputing in India
2. /home/rishabh/workspace/E0236/
Title: Supercomputing in India
3. /home/rishabh/workspace/E0236/
Title: Supercomputing in India
4. /home/rishabh/workspace/E0236/
Title: EKA (supercomputer)
5. /home/rishabh/workspace/E0236/
Title: EKA (supercomputer)
6. /home/rishabh/workspace/E0236/
Title: SAGA-220
7. /home/rishabh/workspace/E0236/
Title: SAGA-220
8. /home/rishabh/workspace/E0236/
Title: Vijay P. Bhatkar
9. /home/rishabh/workspace/E0236/
Title: Vijay P. Bhatkar
10. /home/rishabh/workspace/E0236
Title: PARAM</td><td>
Searching [Java Application] /usr/lib/jvr
Enter query:
Supercomputer in India
Searching for: supercomput india
46893 total matching documents
1. /home/rishabh/workspace/E0236/c
Title: Supercomputing in India
2. /home/rishabh/workspace/E0236/c
Title: Supercomputing in India
3. /home/rishabh/workspace/E0236/c
Title: Supercomputing in India
4. /home/rishabh/workspace/E0236/c
Title: EKA (supercomputer)
5. /home/rishabh/workspace/E0236/c
Title: EKA (supercomputer)
6. /home/rishabh/workspace/E0236/c
Title: PARAM
7. /home/rishabh/workspace/E0236/c
Title: PARAM
8. /home/rishabh/workspace/E0236/c
Title: SAGA-220
9. /home/rishabh/workspace/E0236/c
Title: SAGA-220
10. /home/rishabh/workspace/E0236/
Title: Vijay P. Bhatkar</td></tr>
<tr><td>
Searching [Java Application] /usr/lib/jv
Enter query:
IISc
Searching for: iisc
2410 total matching documents
1. /home/rishabh/workspace/E0236/
Title: National Centre for Sci
2. /home/rishabh/workspace/E0236/
Title: IIT Council
3. /home/rishabh/workspace/E0236/
Title: IIT Council
4. /home/rishabh/workspace/E0236/
Title: E. S. Dwarakadasa
5. /home/rishabh/workspace/E0236/
Title: E. S. Dwarakadasa
6. /home/rishabh/workspace/E0236/
Title: Dipankar Das Sarma
7. /home/rishabh/workspace/E0236/
Title: Dipankar Das Sarma
8. /home/rishabh/workspace/E0236/
Title: Dipankar Das Sarma
9. /home/rishabh/workspace/E0236/
Title: Dipankar Das Sarma
10. /home/rishabh/workspace/E0236
Title: Debasish Ghose</td><td>
Searching [Java Application] /usr/lib/jvm
Enter query:
IISc
Searching for: iisc
2410 total matching documents
1. /home/rishabh/workspace/E0236/c
Title: National Centre for Scie
2. /home/rishabh/workspace/E0236/c
Title: Dipankar Das Sarma
3. /home/rishabh/workspace/E0236/c
Title: Dipankar Das Sarma
4. /home/rishabh/workspace/E0236/c
Title: Dipankar Das Sarma
5. /home/rishabh/workspace/E0236/c
Title: Dipankar Das Sarma
6. /home/rishabh/workspace/E0236/c
Title: Dipankar Das Sarma
7. /home/rishabh/workspace/E0236/c
Title: Dipankar Das Sarma
8. /home/rishabh/workspace/E0236/c
Title: Dipankar Das Sarma
9. /home/rishabh/workspace/E0236/c
Title: Dipankar Das Sarma
10. /home/rishabh/workspace/E0236/
Title: Dipankar Das Sarma</td></tr>
</table>

It is quite clear from the table that BM25Similarity assigns higher rank to more relevant documents as compared to ClassicSimilarity. Hence ranking mechanism of BM25Similarity is better than the ClassicSimilarity.
Ex. For query "IISc" ClassicSimilarity assigns rank 10 to the document titled "PARAM"(name of India's first supercomputer) while BM25Similarity assigns Rank 6 to the same document. Clearly BM25 performed better than ClassicSimilarity in this case.

4 Observations

In this assignment, I have used a tokenizer and various filters from **Lucene** library to analyze the document corpus. I have observed several variations in vocabulary size(in number of terms), index file size(in MB) and posting list size(in MB) when I applied different filters during the analysis. My document corpus consists of ~51k docs of size ~450 MB. Below is the table of observations.

Table: Observations			
Filters	Vocabulary Size(terms)	Index Size(MB)	Posting list Size(MB)
N	435212	142.7	53.5
N+SF	435179	127.8	49.0
N+ST	405223	135.8	55.8
N+SF+ST	405219	122.1	53.0

N-Normalization, **SF**-StopFilter, **ST**-Stemming

In my implementation I have used Default set of Stop Words provided by Standanrd Analyzer (**ENGLISH_STOP_WORDS_SET**).

Some important observations:

1. The size of Index File reduces on applying various filters.

2. There is a reduction of exactly 33 terms when i applied StopFilter on Normalized text(i.e N $\rightarrow$ (N+SF)).
   |**ENGLISH_STOP_WORDS_SET**| = 33

5 Conclusion

In this assignment, I have learned how to use **Lucene** library to build **Searcher** and **Indexer** for an IR system with most basic functionality. It helped me to understand how various filters (e.g. StopFilter, PorterStemFilter, LoweCaseFilter) and ranking mechanism optimize the search. **Lucene** is very intuitive and handy library for building an IR system.